Journal of Simulation (2011) 5, 77–88

www.palgrave-journals.com/jos/

Towards collaborative component-based modelling

HS Sarjoughian[1]*, JJ Nutaro[2] and G Joshi[1]

[1]*Arizona State University, Tempe, AZ, USA; and* [2]*Oak Ridge National Lab, Oak Ridge, TN, USA*

Collaborative modelling enables dispersed users to develop component-based system models in group settings. A realization of such an approach requires coordinating and maintaining the causality of the users' activities. We propose the Collaborative DEVS Modelling (CDM) approach and its realization based on the Computer Supported Collaborative Work (CSCW) and the Discrete Event System Specification (DEVS) concepts and technologies. The CSCW concepts are introduced into the DEVS modelling framework in order to support model development in virtual team settings. A set of modelling rules and tasks enabling collaborative, visual, and persistent model construction and synthesis is developed. To support separate groups of modellers to independently develop models, the realization of CDM supports independent modelling sessions. An illustrative example is developed to demonstrate collaborative and incremental model development. The design of the CDM realization and future research are briefly described.
Journal of Simulation (2011) **5,** 77–88. doi:10.1057/jos.2010.5; published online 14 May 2010

Keywords: collaborative modelling; computer supported collaborative work; discrete event system specification

1. Introduction

Modelling is instrumental to analysis, design, deployment, and evolution of systems that have complex structures and behaviours. Visual and persistent modelling support is considered useful as they significantly simplify common model development activities such as defining a system's parts and their relationships (Rhee, 1999; Sarjoughian *et al*, 1999a; Burmester *et al*, 2005). An important limitation of many of today's modelling approaches and tools is that they are historically developed primarily for single users. To support a team of modellers, it is important to account for space and time constraints on geographically dispersed collaborators. Thus, a collaborative modelling environment should intrinsically support a group of analysts, designers, and system engineers to collectively develop models of systems (Bidarra *et al*, 2002; Delinchant *et al*, 2004; Filho *et al*, 2004). As a team, the modellers can develop, for example, a system-level model of a vehicle that includes engine, controller, and sensor components using the Discrete Event System Specification (DEVS) modelling framework (Zeigler *et al*, 2000). Based on groupware concepts, a collaborative modelling environment can support creating, modifying, viewing, and sharing of the vehicle system model and its components.

Collaboration among modellers can be fostered using Computer Supported Collaborative Work (CSCW) principles and concepts such as sessions, awareness, privacy, and control of individual and group activities (Grudin, 1994; Usability, 2004). CSCW-based environments have been developed to enable multiple users to work as a team under different time and space constraints. A team, for example, can be a group of individuals who are located in different but predictable (or discoverable) places while collaborating in different but predictable time periods. The knowledge and information exchanges can also range from being unstructured (eg unscripted text messaging) to structured (eg orderly creation of a hierarchical model design). Given such alternative collaboration settings, users can manipulate text and documents through blackboard or other web technologies, and further improve their productivity through voice and video-conferencing (Subramanian *et al*, 1999). CSCW is necessary but insufficient to support developing simulation models in collaborative settings (Taylor, 2001). For example, the concepts and methods that have been developed for supporting group activities do not account for the rules under which hierarchical DEVS simulation models can be specified.

In this article, the Collaborative DEVS Modelling (CDM) approach is proposed. The concept of the CDM is realized through extending the component-based DEVS system modelling with groupware support. Therefore, the resulting CDM environment can be viewed to be a combination of the DEVS modelling engine and the Collaborative Distributed Network System (CDNS) engine (Sarjoughian *et al*, 1999b). The *modelling engine* provides modelling constructs (ie creating components, ports, and couplings according to the DEVS formalism). It also supports visual model development and persistence. These capabilities are very useful not only for single-user modelling tools, but even more so in

*Correspondence: HS Sarjoughian, Computer Science and Engineering, Arizona Center for Integrative M&S, Arizona State University, Tempe, AZ 85258-8809, USA.

collaborative settings. The *collaborative engine* supports team-oriented capabilities such as modellers creating or joining collaborative model sessions. Capabilities such as concurrent control, data exchange, and data persistence are used for creating one or more collaborative modelling sessions. This requires maintaining logical ordering among the activities of multiple users since it is impractical for any two users to simultaneously modify or create a model. Collaborative modelling activities must be processed in some well-defined order. For example, a model component can be removed only after it is created. Similarly, concurrent activities of modellers may not be inhibited if they are independent (eg one modeller is viewing a hierarchical model while another modeller is creating a component which is to be added later to the hierarchical model).

The remainder of this article is organized as follows. In Section 2, we review the essentials of CSCW, a selection of related research, and the DEVS modelling approach. In Section 3, we introduce the modelling concepts in the context of a collaborative model development process and devise our approach for DEVS modelling approach. In Section 4, we describe the details of the CDM, an example model, and a sketch of the design of the CDM environment. Finally, in Section 5, we present a summary and discuss open problems and future research directions.

2. Background

It is common to develop models iteratively, especially for systems that are large and complex. System development activities, including model conceptualization and specification, increasingly depend on multiple analysts, designers, and coders collaborating with one another (Filho *et al*, 2004; AnyLogic, 2008). In this section, we briefly review some collaborative environments and their capabilities that have been developed to support better use of information systems and modelling tools. We then turn to the combined role of the system-theoretic and collaborative model development concepts that are used in devising the CDM approach.

2.1. Computer supported collaborative work

CSCW capabilities are universal in that they are intended for a variety of application domains including scientific research in astronomy, bioinformatics, software engineering, and medicine. Based on the key benefits promised by CSCW and pre-packaged software tools, a variety of collaborative environments have emerged in recent years to support team-oriented business and engineering needs (Hill *et al*, 1994; Sun *et al*, 1998; Subramanian *et al*, 1999; Xia *et al*, 2001; Yang *et al*, 2001; Zhang *et al*, 2002). To enable use of single-user software applications in collaborative settings, products such as CodeBeamer (CodeBeamer, 2004) are also developed to support or otherwise augment information sharing and communications. These environments offer collaborative features such as the exchange and creation of information, access to global and local data, and joint use of software applications. For example, AnyLogic (AnyLogic, 2008) uses a version control system and thus can support partitioning a project into parts. Subsequently different users can develop models and carry on related development activities in a collaborative workspace.

Research in the development of collaborative tools has focused on general concepts and communication and control approaches as well as their extensions to specific domains such as process engineering, scientific investigations, and remote health-care diagnosis and planning (Rhee, 1999; Sarjoughian *et al*, 1999b; Subramanian *et al*, 1999; Yang *et al*, 2001). Each of these approaches supports some of the collaboratory concepts (eg shared workspace) and principles (eg causality preservation of collaborators' actions). However, since CSCW capabilities and features are generic, they need to be extended and complemented given the specific needs of the collaborative applications. A recent survey shows that CSCW applications are, for example, in decision support, sharing information, and conferencing with little effort in direct support of constructing and synthesizing simulation models (Jacovi *et al*, 2006). To understand the use of the CSCW concepts and technology, we reviewed some approaches and tools that are developed for specific application domains (Grundy *et al*, 1998; Subramanian *et al*, 1999; Kim *et al*, 2001; Bidarra *et al*, 2002; Filho *et al*, 2004; CanyonBlue, 2005). A detailed review of these can be found in Joshi (2004). The following brief discussion for three of these collaborative environments is intended to highlight the use and adaptation of CSCW concepts and technologies.

Co-Surgeon (Kim *et al*, 2001) is a collaboratory supporting surgical simulation for 3-D anatomical models using a client-server architecture style. The server maintains the database consisting of medical images, patient records, treatment procedures, and surgical plans. To maintain model consistency, activities of participants, the server uses a token-control mechanism in order to allow only one user at a time to control the global view of the model. To obtain control of the session, a participant must first obtain a token. This request is placed in a FIFO queue, which is maintained by the session supervisor. The participants can collaborate synchronously or asynchronously. In synchronous collaboration the session initiator prepares and uploads a model to the server for a prescheduled meeting. Collaborators then obtain a local copy of the model from the server. When a client changes the current view of the model, the server broadcasts those changes to all other clients. There are three basic types of model manipulation: selection, translation, and rotation. To support surgical simulation, operations such as marking, measuring, and cutting are supported.

Collaborative Computer Aided Design (CCAD) systems enable specialists to work collaboratively on the design and development of mechanical and electrical systems (Bidarra

et al, 2002). Because of the computational intensity of CAD, the architecture of CCAD may use thin- or thick-client/server design. A thin-client has a local view of the model located on the server while a thick-client has its own local copy of the model with the original model on the server. CCAD also allows various techniques, such as token control or locking, to manage concurrency. A token control mechanism grants model ownership to only a single client who can modify the model while others are only allowed to view the model. In addition, locking may also be used to provide greater user-control. Depending on the granularity (eg deep *versus* shallow) of the lock, different levels of collaboration are possible. For example, locking can be used to restrict the manipulation of the entire or part of a model to a single client.

GroupSim is a collaborative environment for modelling discrete event simulation systems (Filho *et al*, 2004). It uses the Activity Cycle Diagram notation for non-hierarchical visual modelling activities and their relationships. This environment uses the GroupPlaces groupware architecture which offers the CSCW workspaces that are used in CDM (Sarjoughian *et al*, 1999a). Unlike CDM, requirements such as access control among multiple concurrent modellers and performance are left as future work. Users can synthesize Java classes of the models via their visual object representations. This environment is proposed to be extended to automatically combine the models that are in the form of Java classes and support a user to start a simulation which can be stopped or resumed by a collaborating user.

The CDM approach described in this article uses the basic CSCW concepts and technologies described above to create a collaborative environment for building DEVS models. The above approaches and tools primarily support collaborative model development similar as in collaborative software development. Furthermore, although models constructed with such approaches can be characterized as discrete-event, there is no direct support for constructing models that conform to systems theory and DEVS in particular. The structural specifications including component ports, couplings, and hierarchical construction are not supported. CDM supports public and independent private workspaces and it is possible for any user to simultaneously participate in multiple private sessions. The client view of a model's block diagram is automatically updated as the model changes by multiple users. The model layout is important for collaborative sessions since it allows all modellers share a common view of the system. The updates to the models are generated and sent in a well-defined order to the clients. Since only updates to the models are necessary, there is a reduced need for network resources which in turn improves the performance of the CDM environment.

2.2. Component-based simulation modelling

Systems theory offers a formal foundation for conceptualizing and specifying modular, hierarchical models (Wymore, 1993; Zeigler *et al*, 2000). It supports characterizing state-based hierarchical structure and behaviour of a system. The structure of a system refers to its parts (atomic or composite), input/output interfaces, and how parts may be composed to form the system. The behaviour of a system refers to how inputs are processed and how the state of a system changes both internally and due to the interactions among system components. Object-orientation concepts and constructs such as abstraction, encapsulation, inheritance, and polymorphism offer capabilities for realizing system-theoretic modelling formalisms and extending their modelling capabilities.

The DEVS is a system-theoretic approach to modelling discrete event systems (Zeigler *et al*, 2000). Its atomic and coupled models are defined to have input and output ports which are the only means by which inputs can be received and outputs can be sent. An atomic model specifies the dynamic behaviour of a system's component in terms of states, inputs, outputs, transition functions, an output function, and a time advance function. All leaf nodes of a hierarchical model are atomic models and these cannot be further decomposed. Hierarchical models are defined in terms of atomic and coupled models. A coupled model is defined to have a finite set of atomic or coupled models and every coupled model conforms to strict hierarchy—that is, a model cannot contain itself anywhere in its hierarchy. All model interactions occur through sending and receiving messages through couplings. A coupled model can have internal and external couplings which are used to send and receive outputs and inputs from its parts. Internal couplings from output ports to input ports capture how the coupled model's components influence one another. External input couplings send messages originating from the parent model to its components. External output couplings send messages originating from within the model components to the parent model. These models adhere to causality and timing constraints—inputs sent to atomic (and coupled) models will take some period of time to be processed and when two models are combined into a coupled model, the inputs sent and outputs received to and from the components must be through the input and output ports of their coupled model.

Environments such as DEVS-Suite (DEVS-Suite, 2008) and MATLAB®/Simulink® (Mathworks, 2002) support model development and execution, viewing the structure of models, and animation of their behaviour. Modelling environments also exist to support visual specification of models for single users (Delinchant *et al*, 2004; Burmester *et al*, 2005; Sarjoughian, 2005; CoSMos, 2010). These are called *object-oriented systems-theoretic* modelling environments. A key benefit of object-oriented systems-theoretic M&S approaches such as DEVS is support for incremental model development which is crucial for synthesizing larger models from smaller models while allowing specializing model components.

3. Collaborative modelling concepts

Development of simulation models can take place in conventional, synchronous, or asynchronous modes. Multiple users could be allowed to create and view models. *Conventional Collaboration* (CC) involves immediate, face-to-face communications and requires all participants to be in the same physical space at the same time. *Synchronous Collaboration* (SC), however, can occur when the participants have agreed on a time to collaborate, but are not necessarily in the same physical place (eg video-conferencing). Unlike the other two modes of collaboration, *Asynchronous Collaboration* (AC) does not require any prior agreements on time or space (eg computer bulletin board). These collaboration modes categorize time and place as (a) same, (b) different but predictable, and (c) different and unpredictable (Grudin, 1994; Filho *et al*, 2004). This smaller category of collaboration modes emphasizes teamwork with respect to collaborators joining and leaving in both predictable and unpredictable time instances. In particular, the collaboration modes are not distinguished with respect to the locations of the collaborators, but instead emphasize the concept of simultaneity. This smaller set of collaboration modes is appropriate given the aim of collaborative model development considered in this article.

Many research and commercial environments have been developed to support different kinds of modelling activities across different domains. Examined from the perspective of enabling formal model specification, environments such as DEVS-Suite and MATLAB®/Simulink® do not have the concepts and support for model development in a collaborative setting; instead these tools and their underlying formulations support single users. They offer support for dispersed modellers in the form of sharing data, models, and internet-based communication. They, however, do not provide the necessary logic that can ensure single-user modelling activities can be carried out in a collaborative setting. For example AnyLogic supports multiple users developing models in a group setting, but there is a lack of built-in modelling rules for constructing models. Since the general-purpose nature of collaborative concepts and technologies, CSCW must be complemented with the needs of specific modelling approaches. Before proceeding further, we first develop a mapping of the CSCW collaboration modes to the phases of a common model development process.

3.1. Model construction and synthesis processes

Modelling and simulation is well known to be inherently an iterative process. A comprehensive process called Federation Development Process (FEDEP) has been developed for large-scale, distributed simulation models (IEEE, 2003). The steps for developing new models are requirements gathering, development planning, conceptual modelling, design, and implementation and testing. A simplified process model based on the Grab-and-Glue framework is also considered (Pidd, 2002; Eldabi *et al*, 2004). This process focuses on web-based model construction and model reuse. The key steps in this framework are Grab-and-Glue processes with the purpose to overcome the limitations of traditional modelling and simulation approaches. The approach considers the time and effort for collecting data, constructing models, and executing simulations. It identifies a variety of limitations and challenges including finding models from repositories, model compatibility and adaptation, and visual modelling. Neither the FEDEP model nor Grab-and-Glue approach takes into account collaborative model development modes as described next. Collaborative process development frameworks have also been proposed for software development. For example, SPEARMINT/XCHIP (Fernández *et al*, 2004) supports a graphical, hypermedia structure approach for model development and documentation. In comparison with this approach, the process proposed below is targeted for simulation model development and in particular accounts for system-theoretic modelling with different collaboration modes.

In view of collaborative model development, a process model shown in Figure 1 is proposed. This process consists of the six phases. The bi-directional arrows show the iterative nature of the model and simulation development process. For all but simple and trivial systems, model development demands a mixture of useful modelling concepts with the ability to map domain knowledge to general-purpose modelling constructs provided by modelling formalisms. The process starts with the *obtain requirements* phase which involves clarifying the concepts and the purpose for which the model is to be developed. For non-trivial modelling efforts, the objectives and the formulation of requirements are carried out in CC mode, and sometimes augmented with other modes such as video-teleconferencing to review and share model description or conceptual drawings. After the requirements have been established, data specific to the problem domain is usually collected in an AC mode by the individual team members. During the *collect data and search for models* phase, information necessary to specify models is gathered from expert consultation to searching for models that are developed by others.

Models are typically specified in two ways: (i) *model construction* and (ii) *model synthesis*. The former is concerned with creating new model components and, the later with creating models from pre-built model components (Sarjoughian *et al*, 1997; Lee *et al*, 1998). Model construction includes *specify* and *verify models* phases (see Figure 1). These phases can take place in SC mode and the other two collaboration modes (CC and AC). Model synthesis includes *simulate* and *validate models* phases (see Figure 1). The model construction and synthesis are not exclusive; rather they are complementary to one another since both model

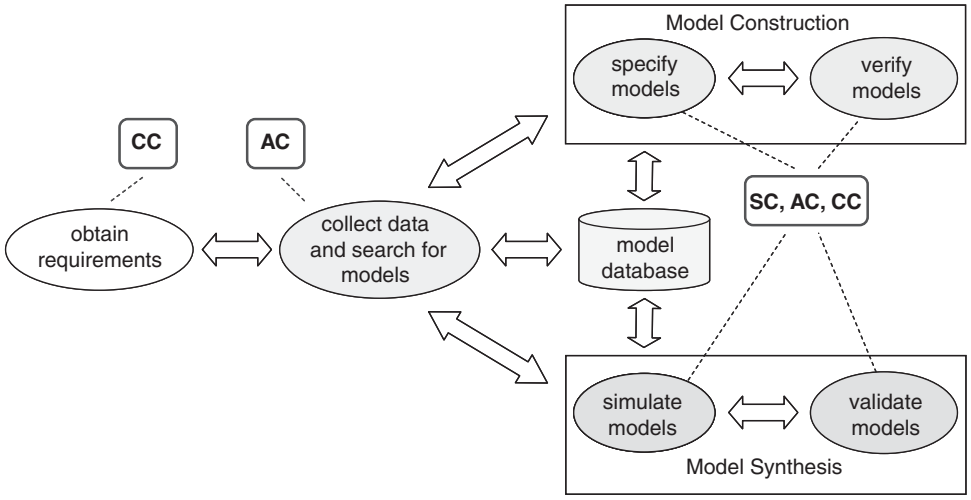

Figure 1 Model construction and synthesis with assigned collaboration modes.

verification and simulation validation are dependent on domain knowledge and simulation experiments. The *obtain requirements* and the *collect data and search models* phases are appropriately tailored to model construction and synthesis.

Model construction and synthesis are iterative since, when specifying or simulating models, it is often necessary to clarify objectives, collect new data and customize or change models that meet the objectives of the simulation study. Since developing large-scale, complex models requires describing many distinct parts of a system and their interactions, not only do modellers need to know how to specify models in a given modelling formalism, but they also must have knowledge of the application domain. Each step in the model construction and model synthesis processes need to be assigned one or more suitable collaboration modes. In Figure 1 the collaboration modes that are considered appropriate for developing simulation models are shown. Although all collaboration modes may be used for any of these steps, some modes are more appropriate than others. For example, a modeller may construct and verify a model for a part of an aircraft and at a later time join a synchronous collaboration session where the aircraft model is synthesized from parts. A user may then simulate the aircraft model and thereafter a group of dispersed users will evaluate the simulation results and validate the model.

3.2. Modelling formalism and collaboration modes

Previously it was noted that existing component-based modelling approaches do not provide concepts and capabilities that can support collaborative model development. A collaborative modelling environment must account for model construction and synthesis which take place over several modelling sessions and in different places. Concurrent model creations and modifications (eg adding a model component or deleting a coupling relation between two model components) among multiple modellers require capabilities that can ensure logical correctness beyond what is assumed for DEVS and other modelling approaches. This requires developing appropriate relationships between the constructs of a modelling formalism and their use in a group setting. The DEVS modelling formalism supports model construction and synthesis activities such as creating a model, adding input ports, and changing coupling between model components.

The ordering of model synthesis steps between two modellers must be well-structured. For example, both modifying a model and viewing the changes to the model must be controlled. The result of removing a model component must be presented consistently to all modellers that are collaborating in synchronous collaboration mode. Likewise, when a component is added to a coupled DEVS model and then coupled to another component, the order of modelling steps must be preserved (ie coupling occurs only after the component is added and the user cannot view the coupling without also viewing the component added to the coupled model). Thus, the CDM collaborative modelling approach must support the DEVS legitimacy property which prohibits any atomic or coupled model to have direct feedback.

Next, the logical DEVS model specification is extended with visual, and persistent models (Sarjoughian, 2005). A logical model refers to the specification of a model's syntax and semantics for a given modelling formalism. A visual model refers to the visual representation of a logical model given the constraints of displaying complete component-based view of models as tree structures or block diagrams. The persistent model refers to storage of logical models. The inclusion of visual and persistent modelling concepts and capabilities with logical modelling are key for collaborative model development.

Modelling formalism:

- The DEVS logical specifications of atomic and coupled model structures must be preserved when used in a collaborative setting. For atomic models, inputs, outputs, states, and functions can be specified once a model component is created. Each modelling step is 'atomic' and no ordering is required among these steps. A coupled model can be created and other models added or removed from it. Flat and hierarchical coupling of models conform to the DEVS modelling formalism.
- Modelling constructs must remain invariant under alternative collaboration modes—that is, collaborative model development may neither weaken nor restrict the DEVS model specifications. If a modeller adds a component to another model component and at some later point the added model is removed by another modeller, both modelling steps are visible to both modellers in the order in which they occurred. Furthermore, there is no guarantee that any modelling step is carried out or viewed at the same wall clock time. All logical modelling steps and their visual representation for atomic and coupled model development must be guaranteed to conform to the DEVS modelling formalism.

Collaboration modes:

- One or more logical model components may be accessed by multiple dispersed users only if the model specifications are consistent with the modelling formalism within which they are described. The creation and modification of model components in synchronous and asynchronous group collaboration must be guaranteed to be consistent with the chosen modelling formalism.
- Every atomic and coupled visual model is a graphical representation of its corresponding logical model. The visual models must remain invariant to time and space. Every dispersed modeller must be assured a correct view of every model as it evolves. That is, the order of logical modelling steps must also be preserved for visual modelling. The order preservation of the modelling steps is independent of network delay and every modeller must view all of the modelling steps whether or not a modelling step is retracted at a later time.
- Logical models, possibly with versioning, must persist across time and space. Models may be stored in alternative media such as flat files or databases. The visual representations of the models are rendered to each modeller separately given one or more repositories of logical models.

The DEVS modelling and CSCW concepts are needed for developing the CDM. A proper mapping of the modelling steps and collaboration modes is required. As noted earlier, model construction and synthesis are complex and iterative which prohibits having more than a handful of modellers to collaboratively develop models. Indeed, at the present time collaborative model construction and synthesis remains challenging to support beyond a handful of people.

4. CDM approach

The development of models among a group of modellers can be realized by enabling system-theoretic modelling capabilities within CSCW workspaces. To support collaborative model development it is important to begin with the activities of a single modeller. His modelling activities need to be formulated in terms of the collaborative session concept and principles. It is also important for concrete realization of the collaborative modelling framework to be simple and efficient.

A *collaborative session* is a loosely bounded workspace within which a group of clients develop a model jointly. Compared to a session in a classical client/server environment in which clients carry out independent tasks, a collaborative session's content is based on interdependent activities of clients. Within a collaborative session, user actions that operate on shared data are synchronized so as to enforce the rules of the modelling constructs. A collaborative session has a finite duration with a start-time and an end-time with the implication that initialization, operation, and termination steps must be supported separately for every modelling session. For example, given a group of modellers, one user first creates a session so that all users can join the session. Then the users, which may not include the person who created the session, can collaboratively develop models. The session remains active until the modelling activity comes to a conclusion (eg the model is frozen to establish a baseline).

A collaborative system-theoretic modelling framework consists of a well-defined modelling approach and a collaborative scheme. For the modelling formalism, the DEVS approach is selected. For collaboration support (eg dispersed users sending and receiving modelling queries/actions), the CDNS (Park, 1998; Sarjoughian *et al*, 1999b), a lightweight distributed computing environment, is selected. The combination of the DEVS and CDNS with collaborative session is the proposed *CDM* (*Collaborative DEVS Modeller*) approach. In the remainder of this section, we describe the details of how the DEVS and CDNS are integrated.

4.1. Modelling activities

The modelling activities for the *specify model* phase are given in Table 1. The activities 1 through 3 differ for the single and group modellers in that these activities are inherently sequential for a single modeller and concurrent for a group of modellers. A common aspect of the activities is to support relatively large-scale model development, especially from the

visualization perspective. This is important for supporting a single modeller and more importantly a group of modellers. Model organization is integral not only in the context of collaborative model development but also for models that have tens to several hundred parts and links (or couplings). As will be described in section 'Visual model representation', special care is needed to support logical and visual modelling among the members of a group.

The CDM environment needs to offer basic capabilities such as loading logical models and manipulating visual models. From the collaborative perspective, it needs to provide the capabilities given in Table 2.

Furthermore, the CDM environment needs to be simple for deployment. The logical and visual representations of the models must be *separated*, but kept *consistent* with one another. This separation is important both in terms of users' ability to develop models and to design an efficient environment. For example, logical models can be stored on servers and visual models can be rendered locally on each modeller's computer which significantly reduces data and transfer frequency.

Logical model specification. The logical representation of the models is defined according to the DEVS formalism. The coupled models have hierarchical tree structure representations. In this article, rather than giving the formal specification of the DEVS, we focus on its structural modelling artefacts and their use. The root of the tree provides the most compact view of the system. Subsequent levels (tree branches) reveal greater detail via decomposition, input/output interfaces, and couplings. The leaf nodes of the tree are atomic components that are not further decomposed. The DEVS logical model specification should enable users to construct hierarchical models under the rules listed in Table 3.

The above rules are extended to concretize the collaborative session concept in terms of the DEVS model development activities. The rules below underscore the importance of modelling activities (eg adding a component or a legitimate link between two model components) in a collaborative session. A modeller can define a coupled model without its parts and another modeller may define one or more parts of the coupled model. Similarly, a model may be defined without ports and only later specified to have ports. The key concept is that collaborative model construction and synthesis imposes requirements that may be unnecessary from the perspective of a single modeller. Modellers in a collaborative session develop models with the understanding that a collaborator can specify a partial atomic or coupled model that may be completed by any modeller. The synthesis of CSCW and DEVS, therefore, provides the basis to account for modelling activities among groups of modellers that span some finite period of time and thus assuring all activities are synchronized to ensure syntactically correct structural logical specification of models. To achieve this, the additional rules provided in Table 4 are defined and supported in CDM.

Given the above, we define the collaborative modelling constructs *add, delete, cut, remove, link, copy*, and *edit*. It is important to note that these logical modelling constructs

Table 1 Model construction and synthesis activities

1.	Logical models can be specified and revised
2.	Visual models represent tree structure and block diagram views
3.	Logical models persist in time and space and can be revised and retrieved

Table 2 Collaborative activities

1.	Enter and exit modelling sessions
2.	Joint and leave modelling sessions
3.	Create, delete, and visualize models

Table 3 Logical model specification rules

1.	The root node of the tree must be a coupled model.
2.	An atomic model does not have any other model component contained within it.
3.	A coupled model can have a finite number of atomic or coupled components; a coupled model cannot contain itself at any level in the model hierarchy.
4.	Atomic and coupled models can have a finite number of input and output ports. At most one link can exist between any two ports that are eligible to be coupled.
5.	A coupled model may have a finite number of links with its components; its components can have a finite number of links with each other. A link is unidirectional and is specified in terms of (source model, output port) and (destination model, input port). The input port of a coupled model can be connected to the input of any of its components. Any output port of any component of the coupled model can be linked to any output port of the coupled model. Any output port of any component of the coupled model can be linked to the input port of any component of the coupled model. A model's output and input ports cannot be coupled to one another.
6.	Each model has a unique identity and its input and output port names are unique among themselves.

Table 4 Additional logical model specification rules for a collaborative session

1.	A coupled model may have no components at a given time during a modelling session. Atomic and coupled models may not have any input and output ports.
2.	A coupled model may have no links at a given time during the modelling session.
3.	A model that is not the root may belong to a coupled model; an atomic model may be defined, but not be part of any coupled model.

satisfy the syntax and semantics of the DEVS models, but in addition, are defined as operations that take place among dispersed modellers and may be carried out at different time instances of a collaborative modelling session. Further details of these modelling actions are defined in the next section.

Visual model representation. Modellers located in dispersed locations can collaborate to build a model of a system; for example, the cabin, navigation, and control components of the aircraft model depicted in Figure 2. To graphically represent such a model, CDM environment provides graphical views for the logical elements (ie *atomic, coupled, input/output ports, links*) of a model described above. The Graphical User Interface supports two complementary views of the same logical model—tree structure and block diagram. The tree structure shows the entire model as a labelled tree and the block diagram shows block representation of one coupled or one atomic model. The block diagram shows ports and couplings of a selected model. By interacting with the tree structure, the user can traverse the model hierarchy and cut or delete any part of a model. The part can be an atomic component or any branch of the tree structure. In the block diagram, a user can add or cut atomic and composite model components, add or cut coupling relationships, add or cut ports, and rename components. Consistency between the alternative views is maintained automatically.

Of these two visual models, it is important to examine the representation of the block diagram. The diagonal block diagram layout is important for individual and collaborative sessions. First, it simplifies visualizing unidirectional links among components when both feedforward and feedback couplings are used in a coupled model. It reduces crossing of the links and simplifies visual complexity of models. The disadvantage is that for large-scale models with sparse couplings, it can be difficult to visualize and work with the models since a relatively large space becomes necessary. Second, the pre-determined block diagram layout is important for collaborative sessions since it is otherwise difficult to guarantee that the modellers have an identical view of the model. The automatic layout is important especially in a collaborative environment where unanticipated changes to the diagram may occur as a result of actions performed by multiple modellers. Moreover, a uniform representation of the block diagram view is important when modellers' interactions are complemented with voice.

Figure 2 shows a simplified hierarchical model of an Aircraft composed of a Cabin and Navigation and Control components. The visual modelling environment is made of left and right panels and bottom and top tool bars. The left panel shows the tree structure of the System (see Figure 2). While each model component is unique, it can appear in multiple places if it is part of a larger model. For example, the coupled Environment component appears in two places subject to the rules defined in section 'Logical model specification'—as a standalone coupled model it appears on the far left side of the tree, and as a component of the System it appears as part of the first branch of the tree structure. The tree structure view does not show port names and couplings. The right panel displays the composite Aircraft model, its parts, input and output ports, and couplings. The models are represented by blocks and are associated with the model component highlighted in the left panel (ie Aircraft shown in Figure 2). The input and output ports for every (atomic and coupled) model is shown in the left-hand and right-hand side of each block, respectively. For example, the Aircraft model has input ports temp and flight and output ports trajectory and cabinPressure.

It is useful to analyse Figure 2 from a *collaborative session* point of view. Let us suppose the Controls model is being developed in San Francisco, the Navigation model in Atlanta, and the Cabin model in Phoenix and Boston. Assuming that each member of the modelling group is developing a subset of the models for the System, the collaborative environment provides an anyplace/anytime workspace local to each modeller, but also supporting a shared workspace for all the modellers. The ability to develop, view, and subsequently modify hierarchical models is essential since all modellers need to work both independently (to specify their models of the system parts) and collaboratively (to ensure their models are synthesized in accordance to a common set of requirements and abstraction of the system).

Figure 2 also shows the three types of coupling that can be specified: (i) external input coupling from the Aircraft model component to the Cabin model component, (ii) internal coupling from the Cabin model component to the Controls model component, and (iii) external output coupling from the Controls model component to the Aircraft model component. The modularity afforded by the modelling approach enables modellers to develop their own models independent of how their collaborators develop theirs. The consequence is that a modeller would be able to devise the components of the Aircraft and how they influence one another through output/input couplings while making the interface (ie input and output ports) of the Aircraft model available to his/her collaborators.

4.2. CDM software architecture

The CDM environment shown in Figure 3 is built based on the CDNS (Park, 1998; Sarjoughian *et al*, 1999b). CDNS provides collaborative session management and basic object exchange capability via Client and Server CDNS modules. It separates *Individual Task*s and *Group Tasks* consistent with the collaborative system-theoretic modelling concepts and the rules defined in Section 4. The Server Modelling Engine is responsible for the Group Tasks: maintaining the master copy of the model, coordinating client access to the model,

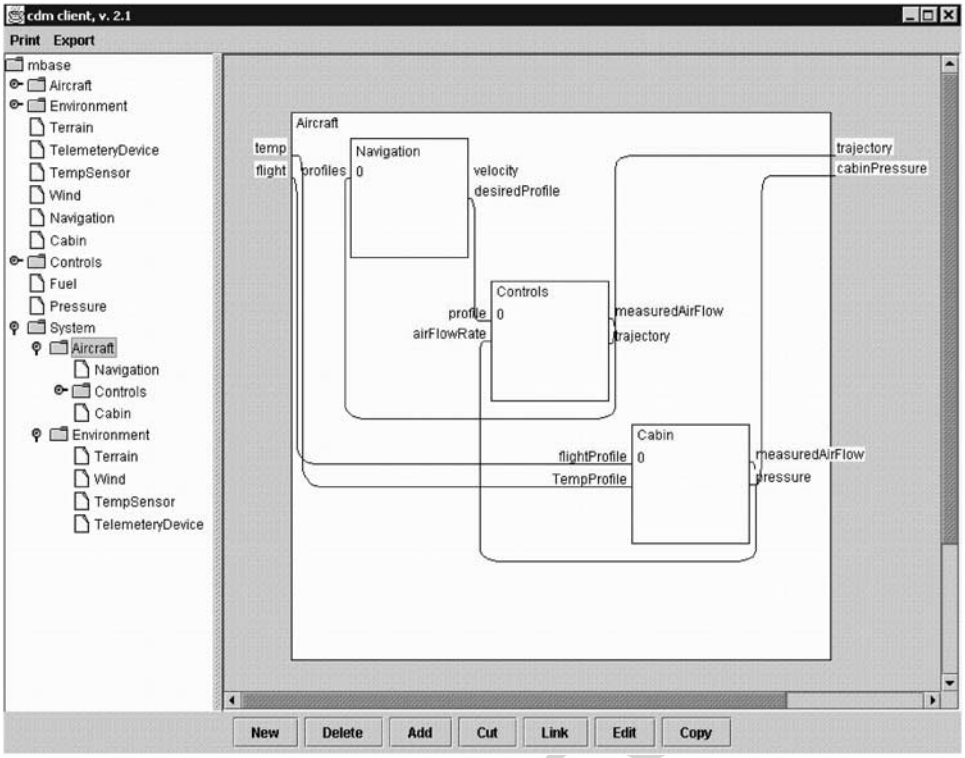

Figure 2 Component representation of the Aircraft model.

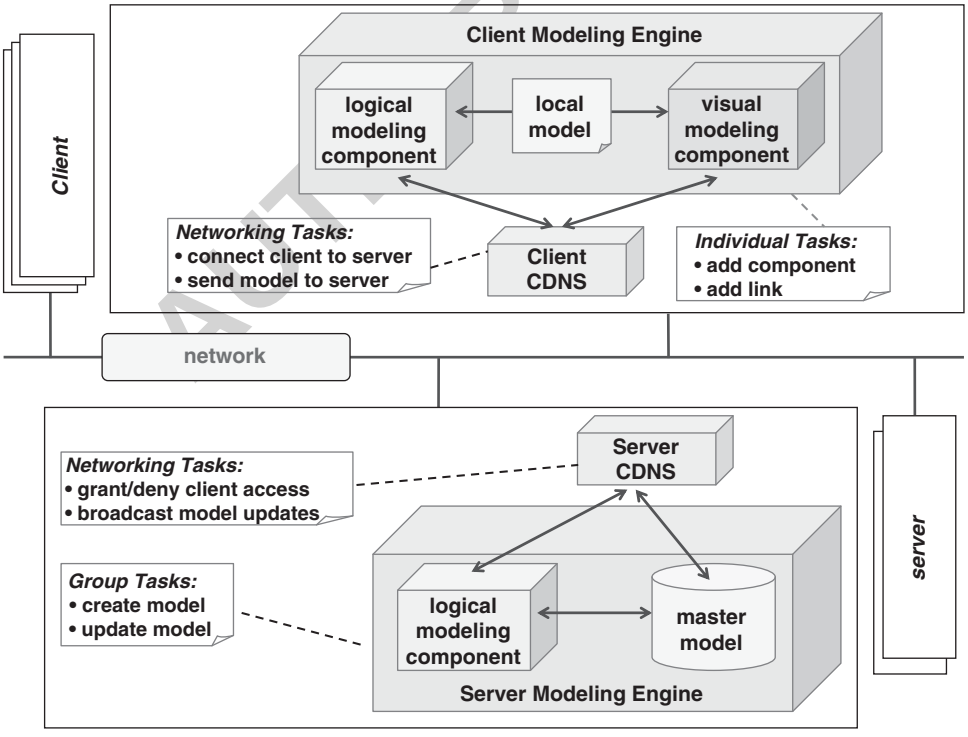

Figure 3 Software architecture for the Collaborative DEVS Modeller.

and informing clients whenever the master model is changed. The Client Modelling Engine is responsible for Individual Tasks; it maintains a local copy of the model that is synchronized with the master copy and is used only for local tasks (eg generating views and performing local consistency checks before sending change requests to the server).

Collaborative distributed network system. The CDNS is a middleware supporting *initiation, termination*, and *interactions* among a set of distributed software applications. This application-neutral middleware, implemented in the Java programming language, provides a flexible foundation for software components (ie Client Modelling Engine and Server Modelling Engine) to send and receive messages transparently. The CDNS architecture is defined in terms of two layers: the *system layer* and the *application layer*. The system layer can support multiple concurrent servers and each server can manage multiple collaborative modelling sessions. Within this environment, a client can be engaged concurrently in two or more different modelling sessions that are hosted by one or more servers. The application layer provides general session-client and session-server services that can be specialized and used for tools such as CDM.

The system and application layers support transparent complex object transmissions (such as trees and lists) through automatic object encoding and decoding that is necessary for distribution across the collaborative workspace. Specifically, it provides a set of primitive operations including *send* and *receive* methods between a client and a server. Examples of such operations would be sending a model from the server to the client. It also provides commands such as *connection request* and *indication object*, where the former can be used to create a modelling session and the latter to confirm its creation.

CDNS provides its own user interface for managing collaborative sessions. Every modeller initially is provided with two windows: *Session Manager* and *System Information*. Before any modeller (client) can request create/delete or join/leave operations, the modeller needs to first *enter* a collaborative workspace hosted by a server. Correspondingly, a modeller can *exit* the collaborative workspace at any time during a collaboration session. These operations are supported by **`Enter`** and **`Exit`** commands from the *Session Manager* window (see Figure 4). Additionally, CDNS also supports **`Create`**, **`Delete`**, **`Join`**, and **`Leave`** operations. The **`Create`** and **`Delete`** operations allow creation of a collaborative session and its deletion. The **`Join`** and **`Leave`** operations enable a modeller to join or leave an existing collaborative session. For example, in Figure 4, `ais1` is a host server offering two collaborative modelling sessions: `Demo1-HostA` and `Demo3-HostA`. Each host session on `ais1`—`Demo1-HostA@ais1` and `Demo3-HoatA@ais1`—has its own model. A prospective modeller may enter `ais1` and upon successful entry, for example, join the `Demo3-HostA` collaborative modelling session. Another modeller can initiate his own collaborative modelling session

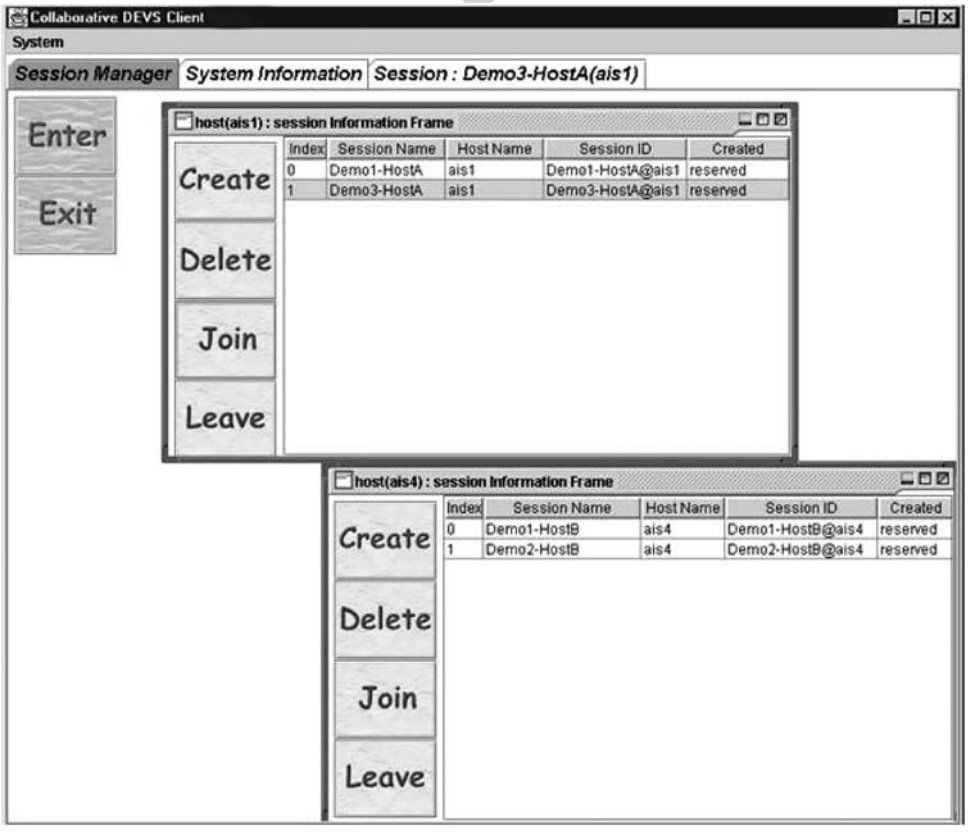

Figure 4 Collaborative modeller workspace for two separate hosts Collaborative Distributed Network System.

(ie `Demo2-HostB`) on the `ais4` host server and join the `Demo1-HostA` collaborative modelling session on `ais1`. Thus, a modeller, who may be hosted by one server can join multiple modelling sessions hosted by another server or multiple other servers. Multiple modellers hosted by different servers may collaboratively develop a model using a standalone modelling session (ie a server owns the model and handles the modellers' development activities).

Client and server modelling engines. The client and server both have modelling engines that communicate using the services provided by CDNS. This architecture is based on the principle of a layered software architecture comprised of generic network services (eg broadcasting a legitimate model change to all members of a collaborative modelling session) and application-specific capabilities (eg adding a port to a model).

The Modelling Engine for the client consists of logical and visual modelling components. The visual modelling component accepts user commands and which are then send to the server. As describe above, the server must verify the user commands are legitimate (eg a model can be added to a coupled model without violating the strict hierarchy constraint). For every legitimate modelling action, as defined in Tables 3 and 4, the local copy of the model is modified and the visual model is updated accordingly. The local model supports visual model manipulation and identifying operations that are legitimate given the local model, but still have to be verified by the server.

A client may change the master model by sending a change request to the server. The server verifies the change against the master model by checking that the modelling rules (see Tables 3 and 4) are not violated. The server then notifies the client as to whether the change was successfully applied to the master copy and, if it was successful, also notifies other clients of the change. The clients, in turn, update their local models as directed by the server. Clients only change their local model when directed to do so.

The Modelling Engine for the server consists of a logical modelling component and a master model. The logical modelling component extends the capability of the client-side logical modelling component by guaranteeing modellers' actions are processed in a well-defined order. The server-side logical modelling component orders modellers' requested actions in a FIFO queue. The requests in every queue are processed one at a time, and those that are legitimate are applied to the master model and pushed to all of clients including the one who requested the operation. Messages from the server to a client use the TCP/IP protocol to ensure reliable, in order delivery. Clients apply change notifications to their local model in the order that they are received from the server.

The only exception to this model update procedure is for clients that are joining a session. A join request is queued by the server in the same way as model change requests. When the join request is processed, the server sends a complete copy of the master model to the joining client. This ensures that new clients begin with a model that is consistent with the master model and the current view of all other clients.

This arrangement allows for synchronous collaboration; each user observes the changes as they occur. If server push updates are removed from this design, only asynchronous collaboration is possible. Whether synchronous or asynchronous, in order for every client to maintain a consistent model, all clients must process operations in the order in which they were received from the server, and message delivery must be reliable. The CDM server guarantees a global ordering of all model actions; this is important to ensure consistent logical and visual views of model modifications among all modellers. The master model has an important role since it supports maintaining consistency among all local models of those modellers who are participating in a modelling session. The server and clients collectively are responsible for maintaining the consistency of all local models with the master model during a collaborative modelling session.

5. Conclusions

We have presented a collaborative system-theoretic modelling environment called CDM, which extends the DEVS modelling concepts and methods for use in group settings. This approach to collaborative modelling introduces a novel alliance between the systems-theoretic modelling paradigm and CSCW for constructing and synthesizing hierarchical component-based simulation models. The realization of the CDM enables developing structural DEVS models in a collaborative setting. It offers modellers the ability to develop structural atomic and coupled DEVS model components. The underlying approach can be extended to support continuous and discrete-time modelling formalisms. The CDM tool may also be used more generally for hierarchical component-based modelling that uses component, ports, and couplings. The modelling environment facilitates multiple, independent modelling sessions, management of modelling actions, and model persistent with complementary tree structure and block diagram model views.

The collaborative modelling approach and its realization provide a basis to handle basic modelling activities that are required in developing simulation models. The underlying collaborative framework supports model creation, management, persistence, and visualization. Since the systems-theoretic approach to modelling dynamical systems is modular and hierarchical, its collaborative realization extends these traits in group settings. The separation of logical and visual models is supported by use of databases which also can simplify creation and use of model libraries. In terms of future research, this environment offers a basis for automatic mapping of the models to their counterpart

simulations that are amenable for simulation. The underlying infrastructure of the CDM can support adding the capability to model behaviours of atomic model components. Such a capability can support specification of state transitions, inputs and outputs, and output functions of atomic models and thus can pave the way to automatically generate executable simulation models.

Acknowledgements—The authors are grateful to the anonymous referees for providing constructive reviews of an earlier version of this article.

References

AnyLogic (2008). XJ Technologies. http://www.xjtek.com/anylogic/, accessed May 2009.

Bidarra R *et al* (2002). A collaborative framework for integrated part and assembly modeling. *J Comput Inform Sci Eng* **2**(4): 256–264.

Burmester S *et al* (2005). Visual model-driven development of software intensive systems: A survey of available techniques and tools. In: *Proceedings of the Workshop on Visual Modeling for Software Intensive Systems (VMSIS) at the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)* Dallas, TX, USA. IEEE Computer Society: Los Alamitos, CA.

CanyonBlue (2005). Enterprise Konesa. http://www.canyonblue.com/products.htm, accessed 30 March 2007.

CodeBeamer (2004). Collaborative software development solutions. http://www.intland.com, accessed 23 April 2007.

CoSMoS (2009). Component-based System Modeler and Simulator. http://sourceforge.net/projects/cosmosim, accessed 15 March 2010.

Delinchant B *et al* (2004). A component-based framework for the composition of simulation software modeling electrical systems. *Simul Trans* **80**(7–8): 347–356.

DEVS-Suite (2008). DEVS-suite simulator. http://devs-suitesim.sourceforge.net/, accessed 30 April 2009.

Eldabi T *et al* (2004). Examining the feasibility of constructing simulation models using the web-based 'grab-and-glue' Framework. In: *Winter Simulation Conference*, Washington DC, ACM, NY.

Fernández A *et al* (2004). Guided support for collaborative modeling, enactment and simulation of software development processes. *Software Process Improve Pract* **9**: 95–106.

Filho WA *et al* (2004). GroupSim: A collaborative environment for discrete event simulation software development for the World Wide Web. *Simul Trans* **80**(6): 257–272.

Grudin J (1994). Computer-supported cooperative work: History and focus. *IEEE Comp* **27**(5): 19–26.

Grundy J *et al* (1998). Serendipity II: A decentralized architecture for software process modeling and enactment. *IEEE Internet Comp* **2**(3): 53–62.

Hill R *et al* (1994). The rendezvous architecture and language for constructing multiuser applications. *ACM Trans Computer-Human Interact* **1**(2): 81–125.

IEEE (2003). *HLA federation development and execution process* Version IEEE 1516.3. IEEE: New York.

Jacovi M *et al* (2006). *The chasms of CSCW: A citation graph analysis of the CSCW conference*. Computer Supported Cooperative Work: Alberta, Canada.

Joshi G (2004). Collaborative component-based modeling using relational databases: Software design and implementation. *Computer Science and Engineering*. Master Thesis, Arizona State University, Tempe, AZ, p 111.

Kim Y *et al* (2001). Collaborative surgical simulation over the Internet. *IEEE Internet Comp* **5**(3): 65–73.

Lee J *et al* (1998). A group-based approach for distributed model construction. In: *31st Hawaii International Conference on System Sciences*, Big Island, HI, USA. IEEE Computer Society: Los Alamitos, CA.

Mathworks (2002). MATLAB. http://www.mathworks.com/, accessed 18 June 2008.

Park S (1998). *Collaborative distributed network system architecture: Design and implementation*. Electrical & Computer Engineering Department, University of Arizona: Tucson, AZ, p 120.

Pidd M (2002). Simulation software and model reuse: A polemic. In: *Winter Simulation Conference*, San Diego, CA: ACM Press, NY.

Rhee I (ed.) (1999). Support for global teams, Guest Editor's Introduction. *IEEE Internet Comput* **3**(2): 30–32.

Sarjoughian HS (2005). A scaleable component-based modeling environment supporting model validation. In: *39th Interservice/Industry Training, Simulation, and Education Conference*, Orlando, FL, USA, IEEE Computer Society: Los Alamitos, CA.

Sarjoughian HS *et al* (1997). Group-enabled DEVS model construction methodology for distributed organizations. In: *11th SPIE*, Orlando, FL. The Society for Modeling and Simulation International, CA.

Sarjoughian HS *et al* (1999a). Collaborative DEVS modeler. In: *Western Simulation Multiconference*, San Francisco, SCS. The Society for Modeling and Simulation International: Alamitos CA.

Sarjoughian HS *et al* (1999b). Collaborative distributed network system: A lightweight middleware supporting collaborative DEVS modeling. *Future Generat Comp Syst* **17**: 89–105.

Subramanian S *et al* (1999). Software architecture for the UARC web-based collaboratory. *IEEE Internet Comput* **3**(2): 46–54.

Sun C *et al* (1998). Achieving convergence, causality, preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans Computer-Human Interacts* **5**(1): 63–108.

Taylor SJ (2001). Netmeeting: A tool for collaborative simulation modeling. *Int J Simul Syst, Sci Technol* **1**(1–2): 59–68.

Usability (2004). First Groupware. http://www.usabilityfirst.com/groupware/, accessed 21 April 2007.

Wymore AW (1993). *Model-based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*. CRC: Boca Raton.

Xia J *et al* (2001). Three-dimensional virtual-reality surgical planning and soft-tissue prediction for orthognathic surgery. *IEEE Trans Inform Technol Biomed* **5**(2): 97–107.

Yang Y *et al* (2001). Real-time cooperative editing on the internet. *IEEE Internet Comput* **4**(3): 18–25.

Zeigler BP *et al* (2000). *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press: New York.

Zhang L *et al* (2002). A feature-based collaborative CAD system. In: *The 7th International Conference on Computer Supported Cooperative Work in Design*, pp 193–197, Rio de Janeiro, Brazil. IEEE Computer Society: Los Alamitos, CA.

Received 29 July 2008;
accepted 9 November 2009